



# Architecture Framework and Viewpoint DSLs Kitalpha

## Development

## Execution

## Process

## Architecture Framework Development

Description

AF DSL

Generation

AF Generation

Packaging

Architecture Framework

## Viewpoint Development

Description

Viewpoint DSL

Generation

Viewpoint  
Generation

Customization

Additional  
Development

Packaging



Metamodels



User Interface



Diagram



Tools

## MBE Workbench

Abstraction Level 1  
viewpointAbstraction Level 2  
viewpointAbstraction Level 3  
viewpointAbstraction Level 4  
viewpointAbstraction Level 5  
viewpoint

Perfo. VP

Safety VP

Cost VP

## Services

Creation of Architecture Framework  
Edition of Architecture Framework  
Generation of Architecture Framework  
Packaging of Architecture Framework

Creation of Viewpoint  
Edition of Viewpoint  
Generation of Viewpoint  
Viewpoint documentation generation  
Packaging of Viewpoint  
Reverse of ecore model to viewpoint dsl

Versioning  
Collaborative work with viewpoints

Scope of



Kitalpha

System description by viewpoints

Viewpoint activation  
Viewpoint deactivation  
Viewpoint detachment  
Viewpoint attachment  
Viewpoint migration

Versioning  
Collaborative work with viewpoints

Reporting  
Architecture Assessment  
Test, Simulation

## Architecture Framework Structure

A set of viewpoints

## Viewpoint Structure

A set of metamodels

A set of rules (check, transformation...)

A set of notations

A set of representations (textual, graphical...)

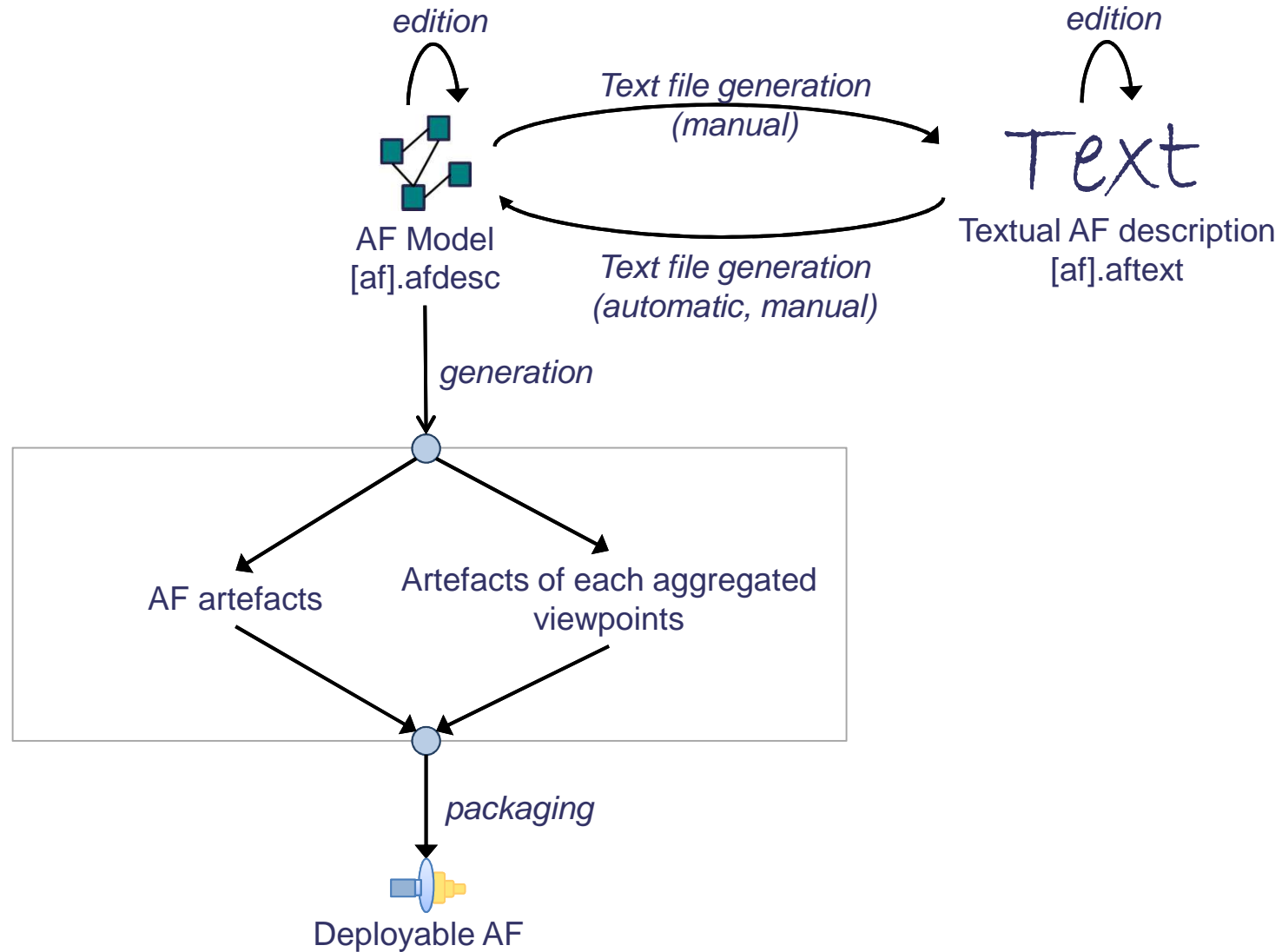
A set of tools

A set of services



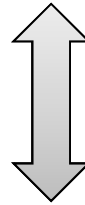
# 1 Architecture Framework DSL

# Architecture Framework Development Lifecycle



**[af].afdesc Model:**

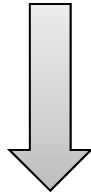
Architecture framework description model,  
independent of any representation (e.g.,  
text, diagram)

**[af]. aftext Model:**

Textual description of architecture

**[af].afdesc Model:**

Architecture framework description model, independent of any representation (e.g., text, diagram)



Right-click on the afdesc file to generate the textual files

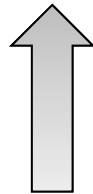
**[af]. aftext Model:**

Textual description of architecture



**[af].afdesc Model:**

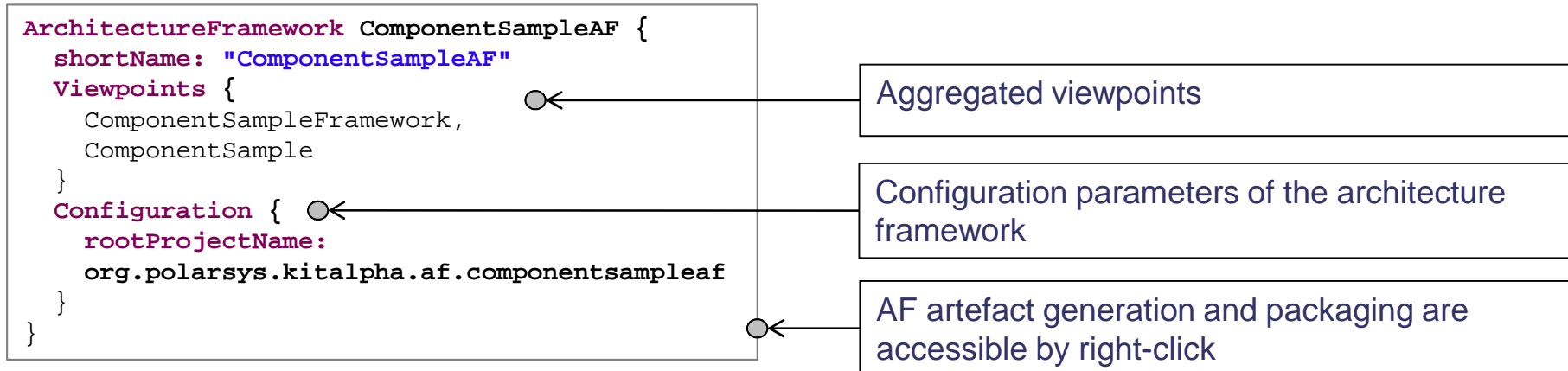
Architecture framework description model, independent of any representation (e.g., text, diagram)



- Automatic on file save
- Right-click on the afdesc file to generate the model

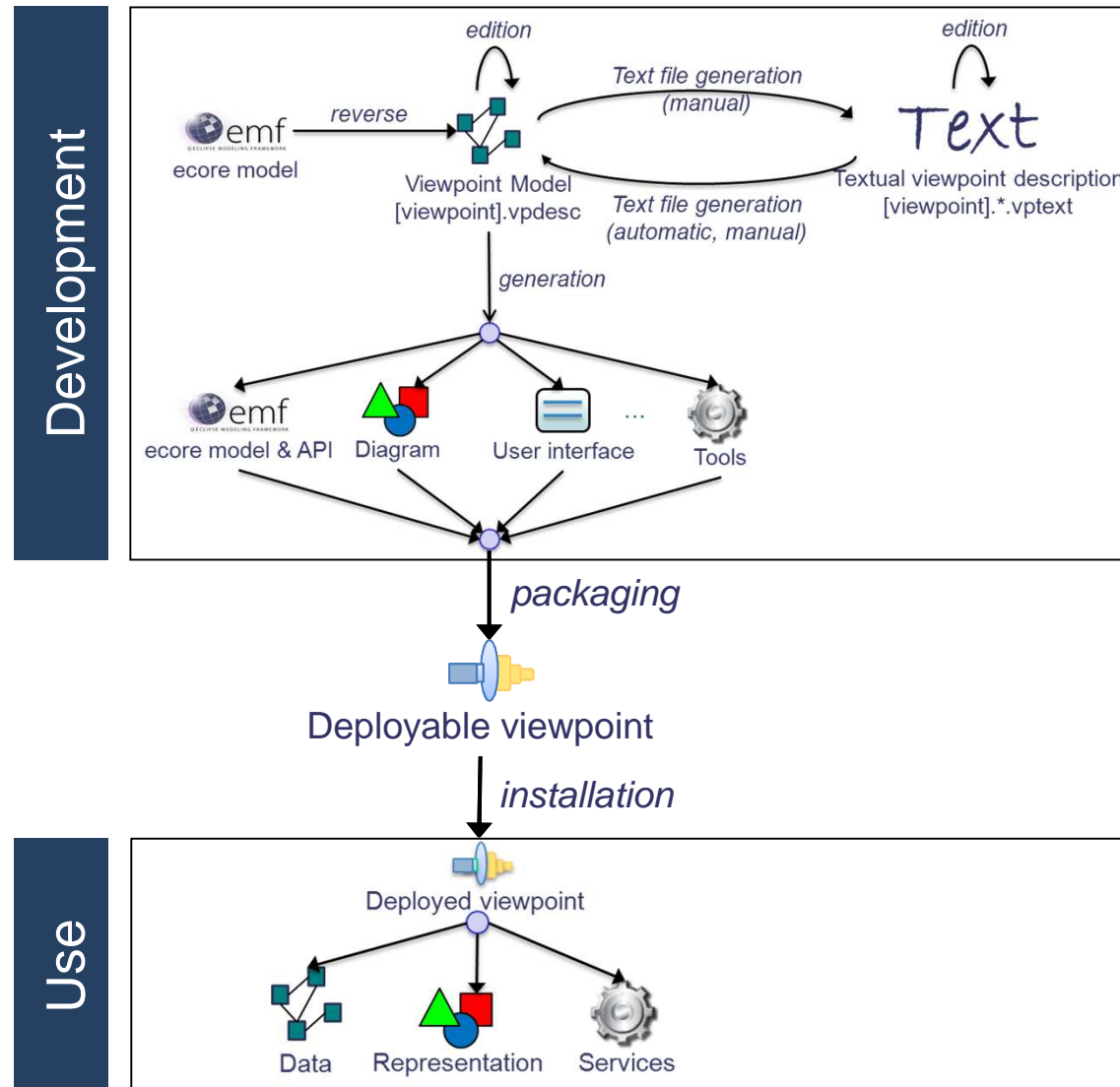
**[af]. aftext Model:**

Textual description of architecture



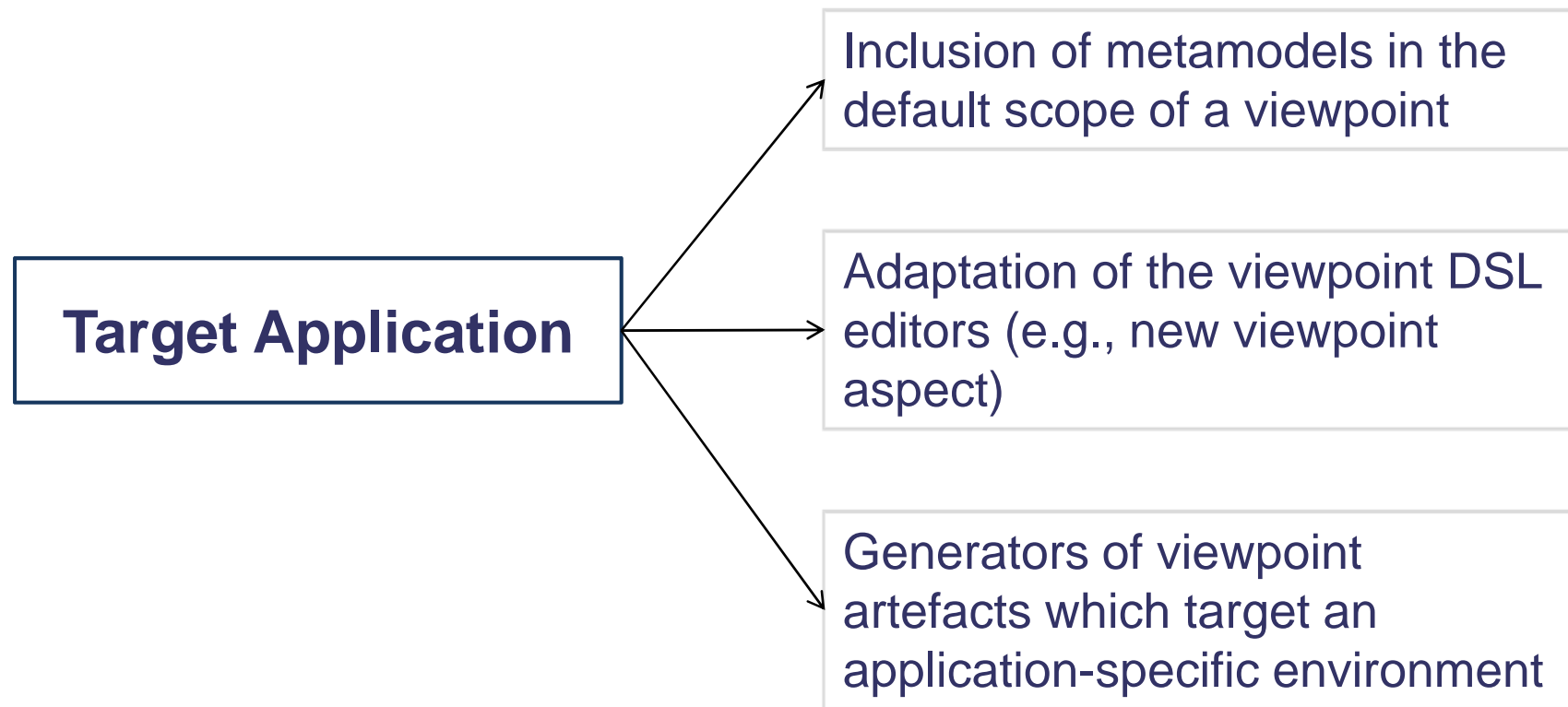


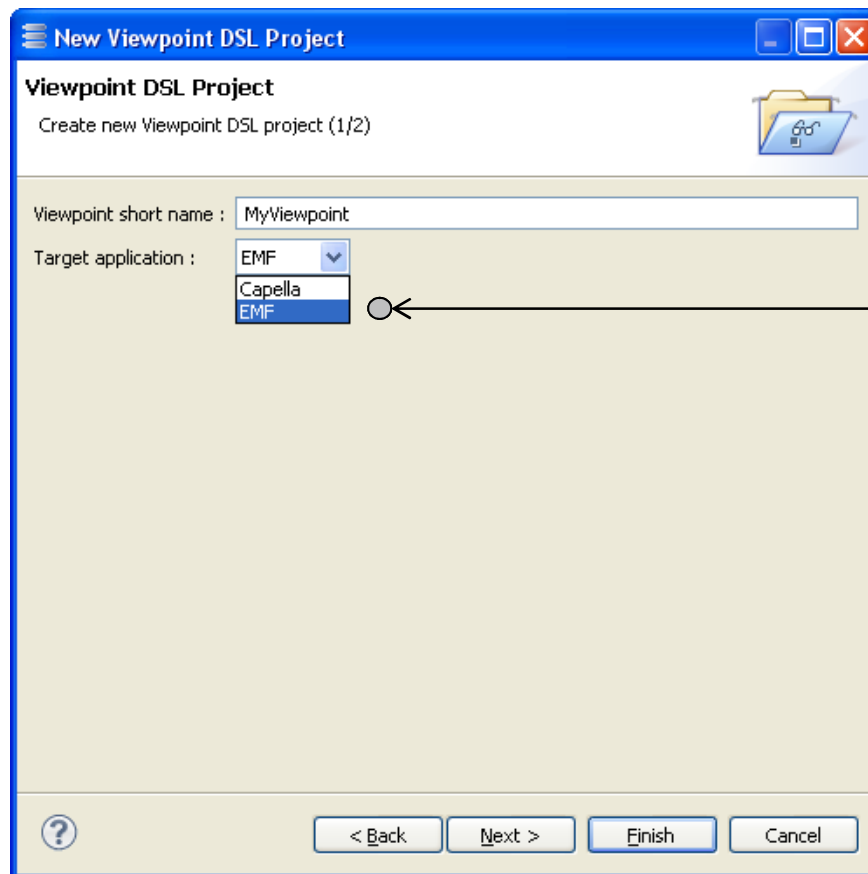
- 1 **Architecture Framework DSL**
- 2 **Viewpoint DSL**



# Viewpoint DSL

## Target Application





At the creation of Kitaha viewpoint project, select a target application.

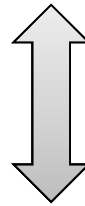
# Viewpoint DSL

## Organization of the editors



**[viewpoint name].vpdesc Model:**

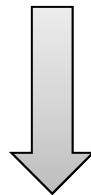
Viewpoint description model independent of any representation (e.g., text, diagram)

**[viewpoint name].[aspect].vptext Model:**

An aspect of viewpoint description (e.g., data, diagram)

**[viewpoint name].vpdesc Model:**

Viewpoint description model independent of any representation (e.g., text, diagram)



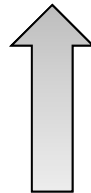
Right-click (viewpoint DSL section) on the vpdesc file to generate the textual files

**[viewpoint name].[aspect].vptext Model:**

An aspect of viewpoint description (e.g., data, diagram)

**[viewpoint name].vpdesc Model:**

Viewpoint description model independent of any representation (e.g., text, diagram)



- Automatic on file save
- Right-click (viewpoint DSL section) on the vpdesc file to generate the model

**[viewpoint name].[aspect].vptext Model:**

An aspect of viewpoint description (e.g., data, diagram)

## [viewpoint name].spec.vptext

Main viewpoint description



# Viewpoint DSL

## Commands and Assistance

## For any editor:

CTRL+SPACE: Text completion

Shift+CTRL+F: Reformat the edited text

CTRL+S: Save the edited text

The outline presents the model equivalence of the edited text

Explicit import declaration is possible just after the file header

## For [viewpoint name].spec.vptext editor:

F3 on a name: Open/Edit a type of viewpoint description

CTRL+SPACE after an editor type: selection of description file

Right-click+ [Generate architecture framework/viewpoint]:

Generation of AF/viewpoint artefacts

Right-click+ [Generate and package AF/viewpoint]: Generation and packaging of AF/viewpoint artefacts

# Viewpoint DSL

## Main Description

```
Viewpoint ComponentSample {  
  name: "ComponentSample"  
  extends ComponentSampleFramework  
  Aggregates Viewpoint1  
  Uses viewpoint Viewpoint2  
  
  Data ComponentSample.data  
  Diagrams ComponentSample.diagram  
  Services ComponentSample.services  
  Build ComponentSample.build  
  Configuration ComponentSample.conf  
}
```

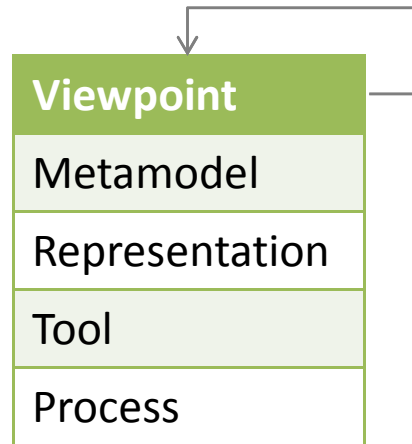
Viewpoint name

Extension, aggregation, use links. Cf. next slides.

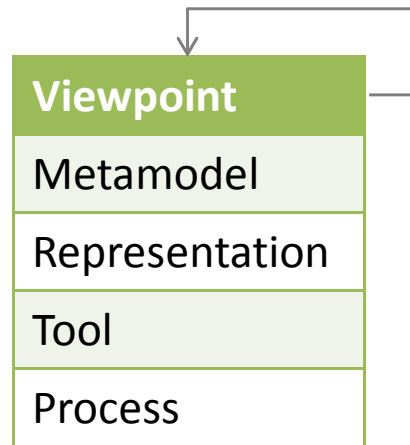
References to the viewpoint configuration data, UI, Diagrams, Services, Build and Configuration description



*use, inheritance, aggregation*

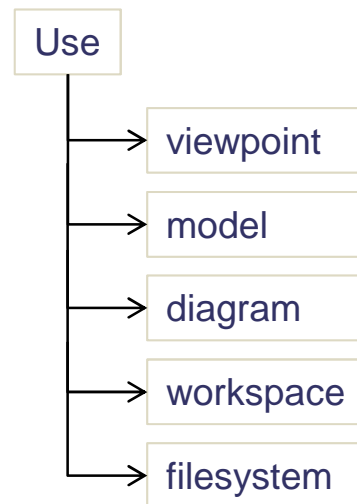


*use, inheritance, aggregation*

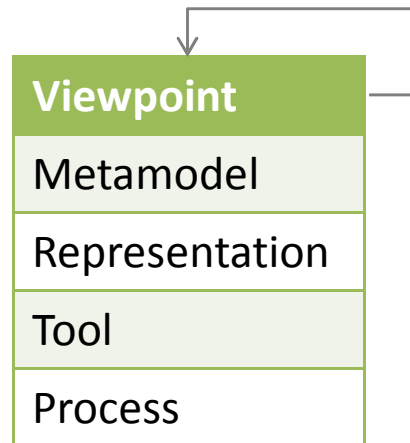


## Use Relationships

Relationship	Meaning
Objective	Enrichment of the viewpoint context by use of resources.



*use, inheritance, aggregation*

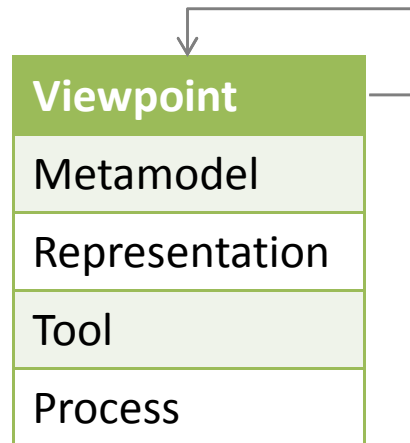


## Use viewpoint

Relationship	Meaning
Objective	Declaration that a viewpoint uses unidirectionally another viewpoint and that its context is enriched by the resources contained by the used viewpoint.
Viewpoint DSL in the main declaration	Viewpoint [name] { uses viewpoint [used viewpoints] }
Viewpoint usage	In the viewpoint DSL, all the viewpoint DSL aspects can access the resources of the used viewpoints thanks to the "import" directive.
Extensibility	For the concerned aspects, a directive enables extensibility. Examples: Data: superClass [list of classes] extends [list of classes] Diagram: DiagramExtension .. { extended-diagram: [diagram reference] }
Packaging impacts	The generated viewpoint plugins have dependencies toward the used viewpoints.

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of Thales. © THALES 2013 – All rights reserved.

*use, inheritance, aggregation*



## Use model

Relationship	Meaning
Objective	Declaration that a viewpoint specifically uses emf-based models (i.e.,.ecore models and instances of.ecore models). The interest is to use a free resource, not contained in a viewpoint.
Viewpoint DSL in the main declaration	Viewpoint [name] { uses model [url of the used Emf resources] }
Viewpoint usage	Like the Use directive
Extensibility	Like the Use directive
Packaging impacts	Like the Use directive

### Declaration in [vp].spec.txt

```

Viewpoint ComponentSampleQualityAssessment {
    name: "ComponentSampleQualityAssessment"
    use viewpoint: ComponentSample

    Data ComponentSampleQualityAssessment.data
    Diagrams ComponentSampleQualityAssessment.diagram
    Configuration ComponentSampleQualityAssessment.conf
}
  
```

### Declaration in [vp].data.txt

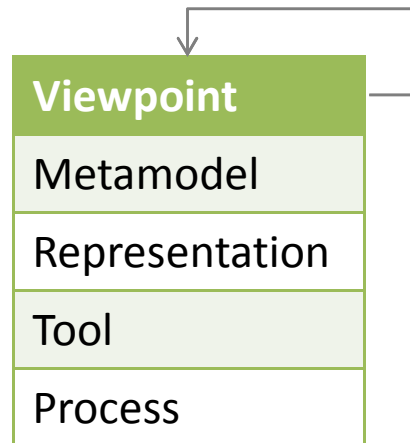
```

import external "http://www.polarsys.org/kitalpha/ComponentSample"

Data ComponentSampleQualityAssessment.data {
    Class QualityAssessment {
        ...
        extends ComponentSample.AbstractComponent
        superClass external ComponentSample.ComponentElement
        ...
    }
}
  
```

[OPEN](#)
**THALES**

use, inheritance, aggregation



## Use diagram

Relationship	Meaning
Objective	Declaration that a viewpoint specifically uses diagrams. The interest is to use a free resource, not contained in a viewpoint. Only Sirius diagrams are recognized yet.
Viewpoint DSL in the main declaration	Viewpoint [name] { uses diagram [url of the used diagram resources] }
Viewpoint usage	Like the Use directive
Extensibility	Like the Use directive
Packaging impacts	Like the Use directive

### Declaration in [vp].spec.txt

```
Viewpoint ComponentSampleQualityAssessment {
    name: "ComponentSampleQualityAssessment"
    use viewpoint: ComponentSample

    Data ComponentSampleQualityAssessment.data
    Diagrams ComponentSampleQualityAssessment.diagram
    Configuration ComponentSampleQualityAssessment.conf
}
```

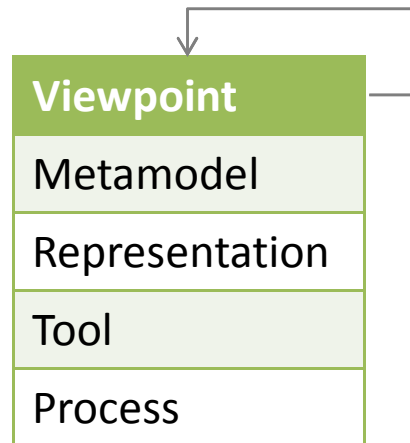
[OPEN](#)

### Declaration in [vp].diagram.txt

```
import external
"platform:/resource/org.polarsys.kitalpha.vp.componentsample.design/description/ComponentSample.odesign"

Diagrams ComponentSampleQualityAssessment.diagram {
    DiagramExtension "ComponentSampleQualityAssessmentDiagram" {
        extended-diagram: ComponentSampleDiagram
        Mapping {
            Container QualityAssessment_ComponentSample {
                import: HardwareComponentContainer_CM Contains {
                    ...
                }
            }
        }
    }
}
```

*use, inheritance, aggregation*

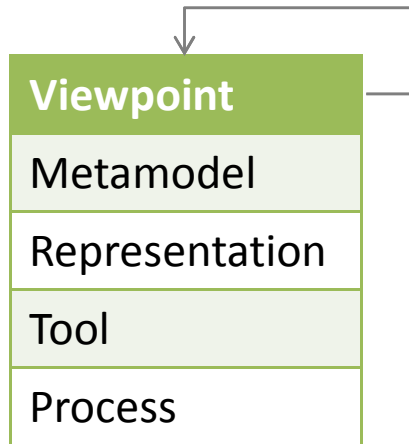


## Use workspace

Relationship	Meaning
Objective	Declaration that a viewpoint specifically uses workspaces. This kind of resource is not exploited yet by a viewpoint aspect.
Viewpoint DSL in the main declaration	Viewpoint [name] { uses workspace [url of the workspaces] }
Viewpoint usage	
Extensibility	
Packaging impacts	

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of Thales. © THALES 2013 – All rights reserved.

*use, inheritance, aggregation*

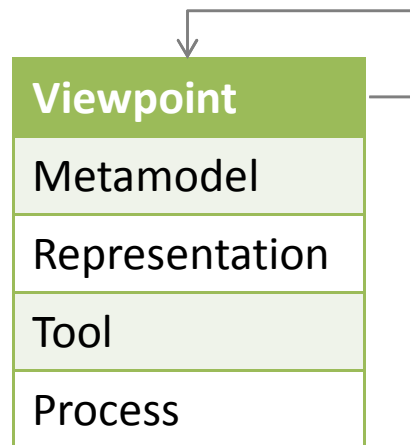


## Use filesystem

Relationship	Meaning
Objective	Declaration that a viewpoint specifically uses filesystem directories. This kind of resource is not exploited yet by a viewpoint aspect.
Viewpoint DSL in the main declaration	Viewpoint [name] { uses filesystem [url of the directories] }
Viewpoint usage	
Extensibility	
Packaging impacts	

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of Thales. © THALES 2013 – All rights reserved.

*use, inheritance, aggregation*



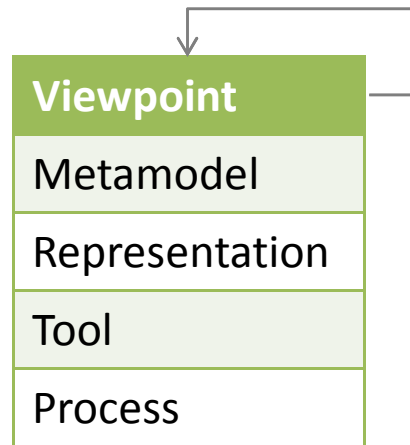
## Inheritance Relationship

Relationship	Meaning
Objective	Declaration that a viewpoint extends another viewpoint.
Viewpoint DSL in the main declaration	Viewpoint [name] {... extends[viewpoint] }
Viewpoint usage	In the Viewpoint DSL, only the Business Rules, Services, and Parameters defined in the Services Viewpoint DSL aspect are managed by the inheritance mechanism like the inheritance in Java. Cf. Viewpoint View accessible by the Viewpoint Manager view.
Extensibility	A Rule, Service, Property redefine an existing one.
Packaging impacts	The generated viewpoint plugins have dependencies toward the used viewpoints.

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of Thales. © THALES 2013 – All rights reserved.



*use, inheritance, aggregation*



## Aggregation Relationship

Relationship	Meaning
Objective	Same principle than the Uses Viewpoint relationship except that the aggregated viewpoints are embedded by the viewpoint.
Viewpoint DSL in the main declaration	Viewpoint [name] { ... aggregates [viewpoints] }
Viewpoint usage	Like the Use directive
Extensibility	Like the Use directive
Packaging impacts	The generated viewpoint plugins includes the aggregated viewpoints.

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of Thales. © THALES 2013 – All rights reserved.

# Viewpoint DSL

## Data

```

Data ComponentSample.data {

  Class ComponentModel { ◉◀
    superClass ComponentElement
    Associations:
      abstractComponent contains [0, *] AbstractComponent
  }

  Class ComponentElement { ◉◀
    superClass external emde.Element
    abstract: true
    Attributes: ◉◀
      name type ecore.EString
      ^description type ecore.EString
  }

  Class AbstractComponent {
    superClass ComponentElement
    abstract: true
    Attributes:
      ^type enum ComponentType
    Associations: ◉◀
      use refers [0, *] AbstractComponent
  }

  Class SoftwareComponent {
    superClass AbstractComponent
  }

  Class HardwareComponent {
    superClass AbstractComponent
    Annotation ◉◀
    "http://org.polarsys.kitalpha/documentation" {
      key: "description" value: "Hardware Component"
    }
    Associations:
      abstractComponent contains [0, *] AbstractComponent
  }

  Enumeration ComponentType {
    Behaviour, ^Data, Service ◉◀
  }
}

```

Class declaration with its name  
Optional: its superclasses, abstract or not

- Declaration of Attributes: {name} [description] type {type} [changeable:, derived:, ordered:, transient:, unique:, volatile: at true or false] [id: at true or false]
- « id: » means that the attribute is used as identifier

- Declaration of Associations: {name} [description] {contains | refers} {cardinality} [external] {class} [changeable:, derived:, ordered:, transient:, unique:, volatile: at true or false] [id: at true or false]
- "external" means that the class comes from an ecore model

Each Class, Association, Attribute, Operation, Enumeration can have Annotations. An annotation has a URL and set of keys / values

Prefixing by "^" or using quotation marks for keywords, such as description or Data.

OPEN

THALES

```

Data ComponentSample.data {

  Class ComponentModel {
    superClass ComponentElement
    Associations:
      abstractComponent contains [0,*] AbstractComponent
  }

  Class ComponentElement {
    superClass external emde.Element
    abstract: true
    Attributes:
      name type ecore.EString
      ^description type ecore.EString
    ...
  }
}

```

- Syntax: external {model}.{class name}
- "external": keyword to refer a Class from an ecore model

- "emde": used for an extension by aggregation. Cf. tutorial on.

```

import external
"http://www.polarsys.org/kitalpha/ad/viewpoint/1.0.0"

Data TutorialViewpoint.data {
  Class ListOfViewpoints {
    Attributes:
      name type ecore.EString
    Associations:
      list contains [0,*] external viewpoint.Viewpoint
  }
}

```

- Declaration of model import
- "external" means that the class comes from another model than the viewpoint

- Example of class reference from an imported model

# Viewpoint DSL Diagram

```

Diagrams ComponentSample.diagram {

Diagram "ComponentSample" {
  domain-context: ComponentSample.data.ComponentModel
  Mapping {

    Container HardwareComponentContainer {
      domain-context: ComponentSample.data.HardwareComponent
      provided-by association ComponentSample.data.HardwareComponent.abstractComponent
      Representation {
        Label { content: ComponentSample.data.ComponentElement.name police: black }
        Style {
          FlatStyle {border: blue background: light_chocolate foreground: light_chocolate }
        }
      }
      Contains {

        Node HardwareComponentNode {
          domain-context: ComponentSample.data.HardwareComponent provided-by association
            ComponentSample.data.HardwareComponent.abstractComponent
          Representation {
            Label { content: ComponentSample.data.ComponentElement.name police: black }
            Style {
              BasicStyle { border-color: black background: light_chocolate form: Square }
            }
          }
        }
      }
    }
  }
}

```

## Diagram Overview

- Description of each **diagram**.  
A diagram has:
  - Context: first metaclass of the diagram description
  - Mapping: graphical representation of the metaclasses with a label and style
  - Actions: actions based on the mapping (creation, delete...)
- Structure of a **mapping**:
  - A container, node and edge mappings
  - A container mapping contains node mappings
  - A node mapping contains borderedNode mappings

- A diagram **context** describes how to start navigating from a context to a container or node, from a container to node, a node to borderedNode
- The navigation can be between metaclasses of the viewpoint or external classes
- The navigation is either declared by an association or query (Acceleo 3)

```

Diagrams ComponentSample.diagram {

Diagram "ComponentSample" {
  domain-context: ComponentSample.data.ComponentModel
  Mapping {

    Container HardwareComponentContainer {
      >○ domain-context: ComponentSample.data.HardwareComponent
        provided-by association ComponentSample.data.HardwareComponent.abstractComponent
      Representation {
        Label { content: ComponentSample.data.ComponentElement.name police: black } <○
        Style {
          FlatStyle {border: blue background: light_chocolate foreground: light_chocolate }
        }
      }
      Contains {

        Node HardwareComponentNode {
          domain-context: ComponentSample.data.HardwareComponent provided-by association
            ComponentSample.data.HardwareComponent.abstractComponent
          Representation {
            Label { content: ComponentSample.data.ComponentElement.name police: black}
            Style {
              BasicStyle { border-color: black background: light_chocolate form: Square }
            }
          }
        }
      }
    }
  }
}

```

## Diagram Mapping

- A **representation** has:
  - A condition for selection between several possible representations: a boolean attribute or Java method which returns a boolean
  - A label
  - A style

- **Label:**
  - Content: string composed of Attribute names, String, and Java method name with returns a string; "+" is used for concatenation
  - Alignment: Center, Left, Right
  - Police Color

### • Style

- *Container* style: 1) image, or 2) FlatStyle with border, background, foreground colors
- *Node* and *borderedNode* style: 1) image, or 2) BasicStyle with border, background colors, and a form (e.g., square, lozenge), 3) HistogramStyle which is a section with a background color and attribute references for min, max and the current values.

## Diagram Edge Mapping

```

Diagrams ComponentSample.diagram {

Diagram "ComponentSample" {
  domain-context: ComponentSample.data.ComponentModel
  Mapping {

    Container HardwareComponentContainer {
      domain-context: ComponentSample.data.HardwareComponent
      provided-by association ComponentSample.data.HardwareComponent.abstractComponent
      Representation {
        Label { content: ComponentSample.data.ComponentElement.name police: black }
        Style {
          FlatStyle {border: blue background: light_chocolate foreground: light_chocolate }
        }
      }
    }
    Container SoftwareComponentContainer { ... }

    Edge HwCtn2HwCtnSwCtnEdge {
      association-context: ComponentSample.data.AbstractComponent.use
      source: HardwareComponentContainer, SoftwareComponentContainer
      target: HardwareComponentContainer, SoftwareComponentContainer
      Representation {
        Style { end-decorator: OutputArrow color: black }
      }
    }
  }
}
...
}

```

- An **Edge** enables to represent model associations. It is described by:
  - The involved association
  - The source and target mapping declarations
  - The representation: begin/end style, color, line style



```

Diagrams ComponentSample.diagram {

Diagram "ComponentSample" {
domain-context: ComponentSample.data.ComponentModel
Mapping {

    Container HardwareComponentContainer {
    ...
    Contains { ...
        Node SoftwareComponentNode {
            domain-context: ComponentSample.data.SoftwareComponent provided-by association
            ComponentSample.data.HardwareComponent.abstractComponent
            Representation { ... }
        }
    }
}
}

Actions{
    Create SoftwareComponentContainerCreation { label: "Software" tool-for:
        SoftwareComponentContainer}
    Delete SoftwareComponentContainerDeletion { tool-for: SoftwareComponentContainer }
    Drop SoftwareComponentContainerDrop { tool-for:
        HardwareComponentContainer.SoftwareComponentNode}
    ReconnectEdge HwCtn2HwCtnEdgeReconnect
        { label: "Reconnect edge example" tool-for: HwCtn2HwCtnSwCtnEdge}
    OpenAction GoAction {label: "Go!" icon: "goAction.gif"}
}
}

```

## Diagram Actions

- Types of **Actions**:
- Create
- Delete
- Drop
- ReconnectEdge
- OpenAction

**Create:** creation of model element identified by the mapping. The label is displayed in the palette. The icon is the one declared in the Data aspect.

**Delete:** suppression of model element identified by the mapping.

**Drop:** move of model element identified by the mapping, for instance from one container to another one.

**ReconnectEdge:** modification of source and target of a relation.

**OpenAction:** execution of Java code accessible from the palette when its precondition is verified. An OpenAction is not related to a mapping. The Java method name is the name of the OpenAction. The icon comes by default from the "icons" directory.

```

Diagrams ComponentSample.diagram {
  Diagram "ComponentSampleDiagram" {◀
    domain-context: ComponentSample.data.ComponentModel
    Mapping {
      Container ComponentPackageContainer {
        ..
        Contains {
          Container HardwareComponentContainer { .. }◀
        }
      }
    }
  }
}

```

```

import external
"platform:/plugin/org.polarsys.kitalpha.vp.componentsample.design/description/ComponentSample.odesign"
Diagrams ComponentSampleQualityAssessment.diagram {
  DiagramExtension "ComponentSampleQualityAssessmentDiagram" {◌
    extended-diagram: ComponentSampleDiagram
    Mapping {
      Container QualityAssessment_ComponentSample {
        import: HardwareComponentContainer_CM Contains { .. }◌
      }
    }

    Node QualityAssessmentNode {
      domain-context: ComponentSampleQualityAssessment.data.QualityAssessment
      provided-by association external emde.ExtensibleElement.ownedExtensions ◌◀
      Representation {
        Label {.. }
      }
    }
  }
}

```

## Diagram Extension

Identification of the diagram to be extended.

Identification of the mapping element to be extended suffixed by "\_CM"

Case when extension of metamodel by aggregation: navigation through the eMDE metamodel. Else, it is a traditional navigation or query.

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of Thales. © THALES 2013 – All rights reserved.

# Viewpoint DSL

## Services

```
Rules ComponentSampleFramework.rules {  
  Rule RuleOne type Java  
  Rule RuleTwo type Java  
  Rule ValidationOne type EMFValidation  
}  
  
Services ComponentSampleFramework.services {  
  Service Service1 orchestrates RuleOne  
  Service Service2 orchestrates RuleOne, RuleTwo  
  Service ValidationService1 orchestrates ValidationOne  
}  
  
Properties ComponentSampleFramework.properties {  
  Property Copyright type String value "PolarSys"  
}
```

- A Rule executes code which conforms to a language type. Only Java and EMF Validation are supported today.
- Syntax: Rule {rule name} type {type language}

- A Service orchestrates the execution of a rule list
- Syntax: Service {service name} orchestrates {rule list}

- Declaration of viewpoint property.
- Syntax: Property {property name} type {Boolean | Float | Integer | String} value {value}

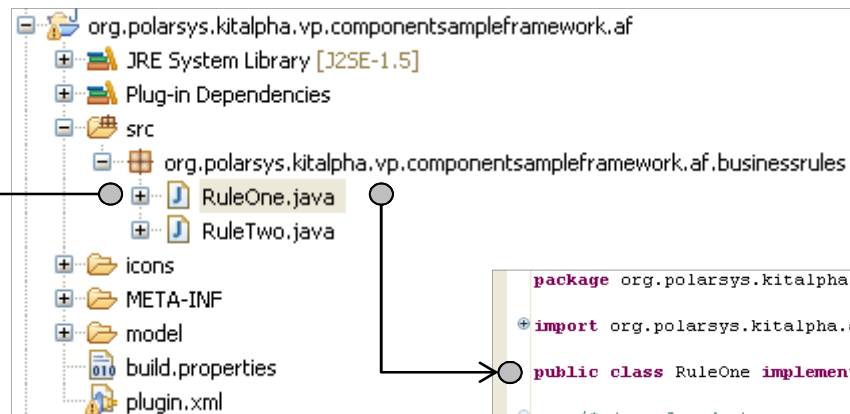
```

Rules ComponentSampleFramework.rules {
  Rule RuleOne type Java
  Rule RuleTwo type Java
  Rule ValidationOne type EMFValidation
}

...
}

```

- During the generation, rules are implemented in the specified language type
- In Java:
  - run method: rule method body
  - canRun method: rule precondition



```

package org.polarsys.kitalpha.vp.componentsampleframework.af.businessrules;

import org.polarsys.kitalpha.ad.viewpoint.integration.rules.JavaRule;

public class RuleOne implements JavaRule {

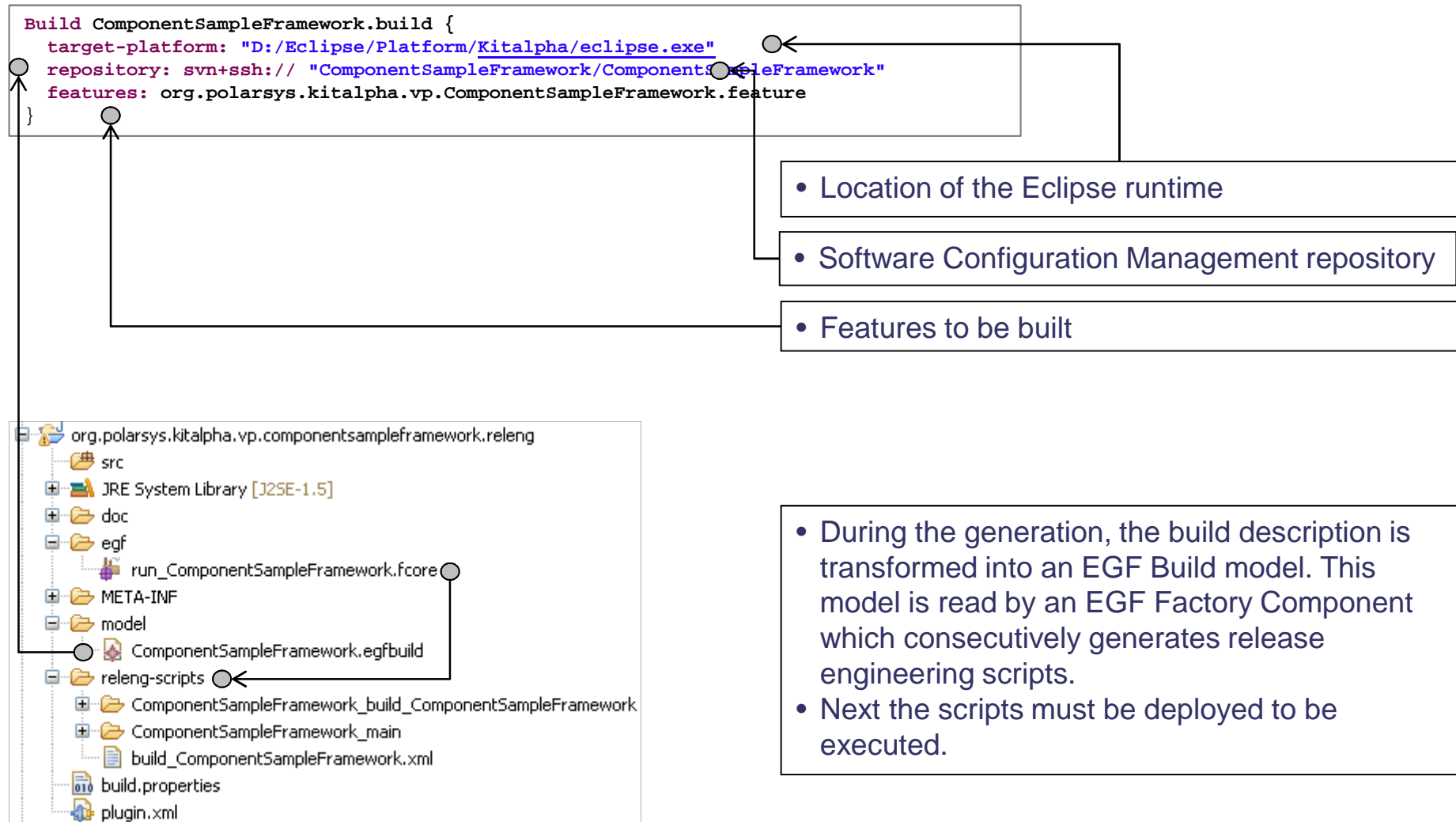
    /* (non-Javadoc)
     * @see org.polarsys.kitalpha.ad.viewpoint.integration.rules.JavaRule#run(org.polarsys.kita.
     */
    public void run(ModelAccessor properties, Object[] selection) {
        // TODO Auto-generated method stub
        org.eclipse.jface.dialogs.MessageDialog.openInformation(
            org.eclipse.ui.PlatformUI.getWorkbench().getActiveWorkbenchWindow().getShell(),
            "Java rule execution",
            "DSL Viewpoint generated rule \n Message from RuleOne rule");
    }

    /* (non-Javadoc)
     * @see org.polarsys.kitalpha.ad.viewpoint.integration.rules.JavaRule#canRun(org.polarsys.kita.
     */
    public boolean canRun(ModelAccessor properties, Object[] selection) {
        // TODO Auto-generated method stub
        return true;
    }
}

```

# Viewpoint DSL

## Build



# Viewpoint DSL

## Configuration



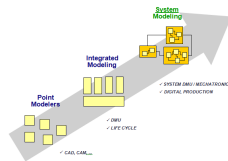
```
Configuration ComponentSampleQualityAssessment.conf {  
  target EMF  
  project org.polarsys.kitalpha.vp.componentsamplequalityassessment  
  nsuri "http://www.polarsys.org/kitalpha/ComponentSampleQualityAssessment"  
  generation {  
    data (  
      Model: true  
      Edit: true  
      Editor: false  
      Test: false  
      Javadoc: false  
      OverwriteEcore: false  
    )  
  }  
  diagram (  
    OverwriteOdesign: false  
  )  
  documentation (  
    EcoreToHtml: false  
  )  
}
```

Mandatory part: the target application, project prefix, namespace URI

EMF generation option: generation or not of model, edit, editor, test API, and the Javadoc. Overwrite enables to reuse an existing ecore model or to keep safe an existing version of an ecore model.

Option to reuse an existing diagram or keep safe an existing version of diagram.

Option to generate or not the html metamodel documentation.



Kitalpha is supported by **Sys2Soft** and **Crystal**, respectively French and European projects



# Th<sub>α</sub>nk You!

<https://www.polarsys.org/projects/polarsys.kitalpha>

[benoit.langlois@thalesgroup.com](mailto:benoit.langlois@thalesgroup.com)

[#LangloisBenoit](#)